A-018

分散相互排除アルゴリズムにおける冗長化管理プロセスを用いた適応型フェイルオーバー

Adaptive Failover with Redundancy Management Processes on The Distributed Mutual Exclusion Algorithm

橋爪 由道 †　　和崎 克己 ††
Yuto Hashizume　Katsumi Wasaki

1　はじめに

本研究は管理プロセスを冗長化した分散相互排除アルゴリズム [1] を形式的仕様記述言語 LNT[2] でモデル化し，CADP toolbox[3] を用いて網羅的検証を行うことで一定の条件下での耐故障性を示すことを目的とする．利用権を制御する管理プロセス群（クオーラム）を二次元コータリーとして構成し，分散相互排除アルゴリズムの実行中に管理プロセスが停止故障した際に必要なフェイルオーバー手法の提案を行う．故障したプロセスのコータリー上の位置に応じ，代替管理プロセスの選定とクオーラムの再構築を行うことで故障プロセスが増えた場合でもできるだけ多くのクオーラム数が取れるようにする．

2　複数管理プロセスによる分散相互排除アルゴリズム

ある共有リソースを利用するリクエストが複数存在する場合にそのうちの 1 つを選ぶ問題を相互排除問題といい，特に分散システム上で相互排除問題を扱う問題を分散相互排除問題という [1].

2.1　複数管理プロセスによる分散相互排除アルゴリズム

以下は相互排除アルゴリズムが満足すべき条件である

- 複数のプロセスが同時に利用権を持つことがないこと
- 永遠に利用権を持たないプロセスが存在しないこと

分散相互排除アルゴリズムは，全プロセスで優先度に関する共通のルールを設定することで条件を満たすことができる．しかし資源を利用するプロセス同士で相互排除を行うアルゴリズムにはいくつか問題がある．そこで，プロセスを資源を要求する“要求プロセス”と資源の利用権を管理する“管理プロセス”に分ける“複数管理プロセスによる分散相互排除アルゴリズム”を作成した．

2.2　分散相互排除におけるコータリーの適用

資源割り当てにコータリー（coterie）という構造を用いる．コータリー C の満足すべき条件は次のように書くことができる [1]．ここで U は管理プロセスの集合である．

- 任意の $Q \in C$ は $Q \subseteq U$ かつ $Q \neq \emptyset$ である．
- 任意の $Q, Q' \in C$ に対して $Q \not\subset Q'$ が成立する．
- 任意の $Q, Q' \in C$ に対して，$Q \cap Q' \neq \emptyset$ が成立する．

要求プロセスはコータリー内の 1 つのクォーラム Q の全要素から許可を得られれば利用権を獲得する．そのため一部の管理プロセスが故障しても要求プロセスは利用権を得ることができる．

2.3　二次元コータリー

コータリーの構成方法として，二次元コータリーがある．ここでは U の要素数 $N = m^2$ であると仮定する．U の各管理プロセスを m 行 m 列の格子状に配置して，クォーラム $Q_{i,j}$ を i 行目と j 行目の要素から成る集合とする．このとき，$Q_{i,j} \cap Q_{i',j'}$ には必ず $S_{i,j'}$ が含まれ，コータリーに関する他の条件も成立する．

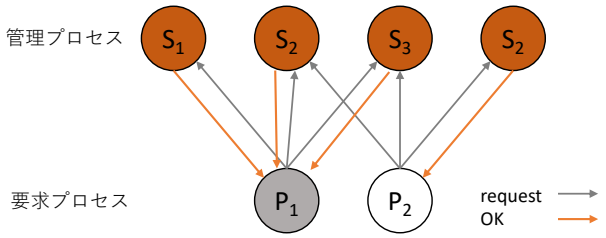

図 1　要求プロセス P_1 が利用権を得るケース

2.4　プロセスの挙動

要求プロセスは 1 つのクォーラムに対して利用権を要求する request メッセージを送信する．管理プロセスは request メッセージを受信すると，自身の持つ優先度付きキューにメッセージを格納する．そして利用権を与える OK メッセージをどの要求プロセスにも送信していない場合，キューの先頭に OK メッセージを送信する．クォーラムに含まれる全ての管理プロセスから OK メッセージを受信し，資源の利用権を得た要求プロセスは資源を利用したのち，許可を得た管理プロセスに利用権を解放する release メッセージを送信する．管理プロセスは release メッセージを受信後，許可を出した要求プロセスの情報をキューから削除する．管理プロセスが要求プロセスに利用権を与えた後に，より優先度の高

† 信州大学大学院総合理工学研究科, Graduate School of Science and Technology, Shinshu University
†† 信州大学工学部電子情報システム工学科, Department of Electrical and Computer Engineering, Faculty of Engineering, Shinshu University

い要求プロセスから request メッセージを受け取った場合，利用権を与えた要求プロセスに対して許可取り消しの cancel メッセージを送信する．要求プロセスがまだ資源を利用していない場合は管理プロセスにそれを承諾する canceled メッセージを送信するが，すでに資源を利用していた場合はそのメッセージを無視する．

2.5 LNT によるモデル化と検証

LNT を用いて図 1 の要求プロセス 2 個・管理プロセス 4 個のモデルを作成する．並列同期を用いネットワークを独立したプロセスとして記述することによりモデル記述を柔軟にする．作成したモデルに対し CADP toolbox を用い，有界公平性に関する仕様「利用権を要求したプロセスは，いつかは利用権が得られる」を確認する．またネットワークと並列に資源を利用している要求プロセス数のカウンタを設置することにより，相互排除が行われていることを確認する．

3 二次元コータリーを用いたフェイルオーバー

二次元コータリーを用いた分散相互排除の問題に，管理プロセスが 2 つ以上故障した場合，コータリーの満足すべき条件を満たせないクォーラムが生じることがある．そこで，管理プロセスが停止故障した際，二次元コータリー上でプロセスの位置を移動させることでできるだけ多くのクォーラムを構成するフェイルオーバーの手法についてモデル化し検証する．

3.1 フェイルオーバーの手法

図 2 の (1) のように行番号と列番号どちらも同じでない 2 つの管理プロセスが故障した場合，両方のプロセスを含む 2 つのクォーラムは条件の一つである「任意の $Q, Q' \in C$ に対して，$Q \cap Q' \neq \emptyset$ が成立する」を満たさず，相互排除を行えない．一方で図 2 の (2) のように，同列あるいは同行に存在するプロセスであれば，故障したとしても全てのクォーラムが条件を満たす．そこで，最初に故障した管理プロセス $S_{0,0}$ を基準とし，新たにプロセス $S_{i,j}$ が故障した際，プロセス $S_{0,0}$ の隣に存在するプロセス $S_{0,1}$ を $S_{i,j}$ の代わりとすることにより，故障プロセスが増えた場合でもできるだけ多くのクォーラムを構成する．

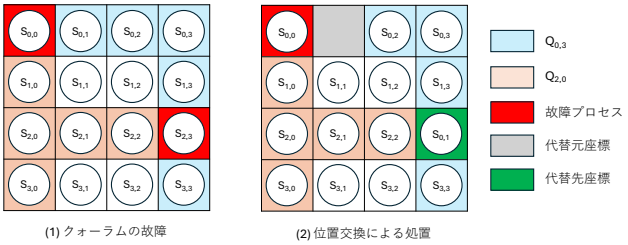

図 2 クォーラムの故障と位置交換による処置

3.2 モデル作成方法

実現にあたり, 管理プロセスは隣接する管理プロセスと情報を交換し，クォーラムの組み替えはスーパーバイザーというプロセスが行う．また状態爆発を避けるため，モデル作成はプロセスの故障数に応じて段階を分けて行う．

■優先度付きキューとプロセスの状態の交換 故障プロセスを正常プロセスが代替する際，故障したタイミングによっては優先度付きキューとプロセスの状態が必要となる．そのため，管理プロセスは二次元コータリー上で隣接するプロセスとそれらの情報を交換する．情報の交換には独立したネットワークプロセスではなく，特別な通信路を用いる．

■スーパーバイザー スーパーバイザーは Listing 1 のようにネットワークプロセスと並列に設置し，要求・管理プロセスと交換した情報によってクォーラムの管理を行う．

Listing 1 並列合成の LNT 記述

```
process MAIN[send, recv, exchange : any] is
  par send, recv, keep_alive, alive in
    par
      par
        requirements_process[send, recv]
      ||
        requirements_process[send, recv]
         ...
      end par
    ||
      par exchange in
        management_process[send, recv,
          exchange]
      ||
        management_process[send, recv,
          exchange]
         ...
      end par
    end par
  ||
    par
       superviser
    ||
       network[send, recv, allwaiting]
    end par
  end par
end process
```

■プロセスの故障数と段階 作成するモデルは管理プロセスが 2 つ以上同時に壊れることはないものとする．また，モデルは管理プロセス $3 * 3$ のモデルであり，故障数に応じて 3 つの段階に分割し，それぞれについて相互排除が行えるか検証する．

1. 1 つ目に故障したプロセスをコータリーから削除する．
2. 2 つ目のプロセス故障に対してクォーラムを組み直す．
3. 3 つ目のプロセス故障により，避けられないクォーラムの故障に基づいた対処を行う．

参考文献

[1] 真鍋義文“情報工学レクチャーシリーズ 分散処理システム”森北出版株式会社,2013.
[2] INRIA/VASY-INRIA/CONVECS, Reference Manual of the LNT to LOTOS Translator(Version 7.2).2023. https://cadp.inria.fr/ftp/publications/cadp/Champelovier-Clerc-Garavel-et-al-10.pdf
[3] CADP http://cadp.inria.fr/